design ideas

Edited by Bill Travis and Anne Watson Swager

Fast algorithm computes histograms

Lukasz Sliwczynski, University of Mining and Metallurgy, Cracow, Poland

A HISTOGRAM IS A convenient method of presenting statistical features of data; you can use a histogram for an estimate of the probability density function of a random phenomenon. To prepare a histogram entails dividing the interval occupied by data values into some number of smaller intervals and then counting the number of occurrences of data in each subinterval. You can easily generate histograms with the aid of standard mathematical packages, such as Matlab. The problem lies in the fact that calculating a histogram using the standard Matlab function hist{} from a large file of data (for example, 10 million data points) with high accuracy (for example, with 12-bit resolution) may take substantial time—many hours, or even days, depending on computer speed and memory. The problem is that hist{} treats its input data as "analog" data with almost infinite precision, and the routine must use program loops, which are very slow in Matlab.

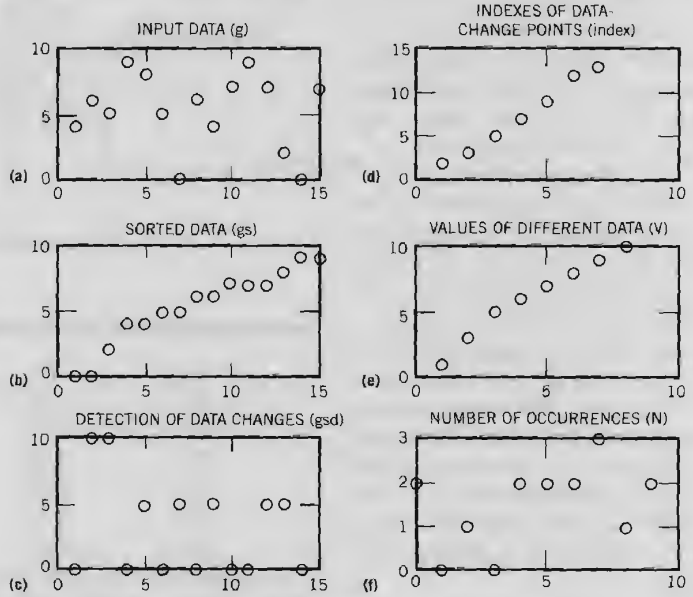
However, you often perform measurements with the use of A/D conversion, with the result that the input data for the histogram-calculating procedure are discrete and that the smallest difference between any two data values is one. This fact allows you to write a special function for calculating the histogram. You can use the complete Matlab program for calculating histograms from discrete data (**Listing 1**). The program is intended for placement into a Matlab m-file, which creates a function from the routine. This function is accessible from the Matlab command window, just as any other function. **Figure 1** illus-

LISTING 1—MATLAB PROGRAM FOR FAST HISTOGRAM CALCULATION

```
function N=fasthist(g,L,W)
%
%     N=fasthist(g,L,W)
%
%     Function calculates histogram of data from D/A converter.
%     Range of numbers is from 0 to 2^L. Subinterval length is W.
%     L is the number of bits of the converter.

g=g(:)';                                                          (1)
gs=sort(g);                                                       (2)
gsd=diff(gs);                                                     (3)
index=find(gsd);                                                  (4)
V=[gs(index),gs(length(gs))]+1;                                   (5)
N=zeros(1,2^L);                                                   (6)
N(V)=[index(1),diff(index),length(gs)-index(length(index))];      (7)
if W~=1                                                           (8)
        for i=1:fix(2^L/W)                                        (9)
            N(i)=sum(N(W*(i-1)+1:W*i));                           (10)
        end;                                                      (11)
end;                                                              (12)
N=N(1:fix(2^L/W));                                                (13)
```

Figure 1



Ordering of the data points in diagrams a through f follows the steps in Listing 1.

trates the operation of the algorithm. The first line (1) of the program prepares input data and gives the data the form of the column vector g (**Figure 1a**). The second line (2) performs data sorting and arranges the data in ascending order (gs in **Figure 1b**). The difference vector gsd in **Figure 1c** has peaks only at those points where data changes; elsewhere, the vector has a value of zero. The distance between two consecutive peaks is equal to the number of data samples of the same values, and the height of a peak is equal to the distance between two consecutive numbers.

Next, the routine (4) calculates the indexes of data changes (**Figure 1d**) and calculates the values of data V (**Figure 1e**) appearing in input vector g (5). A "1" in this code line is necessary because a vector index cannot equal zero in Matlab. In the following step (7), the step vector V indexes the output vector from the procedure N. This indexing operation is simple and fast because both indexes and data are natural numbers and belong to the same interval $<0;2^L-1>$. At this point, the essential part of the procedure is finished. The numbers N(i) (**Figure 1f**) are equal to the total number of occurrences of data with the value i−1 in the input vector g. The last six lines of the program reduce the initial resolution of the A/D converter that you use to acquire data. To obtain faster execution of the program, you switch on this part of the routine only when necessary—for example, when W does not equal 1.

The effect of using this algorithm rather than the standard one in the Matlab package is significant. Using the standard function hist{}, the time to compute a histogram from 10,000 points is 10 sec; for the same data file with 12-bit resolution, the algorithm fasthist{} requires only 0.55 sec. These figures are based on using a 100-MHz Pentium machine with 33 Mbytes of RAM, running under Windows 95. Using fasthist{}, a histogram from a 10 million-point data file takes approximately a half-hour as opposed to more than a day using hist{}. These results occur using Matlab version 4.2. Version 5.2 gives somewhat different results: 68 sec for hist{} and 0.38 sec for fasthist{}, a 179-times reduction. (DI #2422)

To Vote For This Design,
Circle No. 358

Loop powers current transmitter

By N Kannan, Mediatronix Ltd, Triandrum, India

In the circuit in **Figure 1**, the loop supply is 20 to 30V dc, and the loop current is 10 to 20 mA. IC_1 operates as a constant-current (I_p) source. IC_2, R_1, and R_2 operate as a shunt regulator, which provides 12V V_{CC}. IC_2 is a TLO32 op amp. The sensor circuit could be a resistance-temperature-detector circuit or any other sensor- or signal-conditioning circuit. The IC_2/Q_1 combination operates as a current sink, with $I_C=V_{IN}/R_E$. The total loop current is I_p+I_C. Because of its common-mode input-range limitation, IC_2 requires a minimum 2V input for proper operation. Hence, the V_{IN} signal range must be 2 to 10V. With $V_{IN}=2V$, you adjust potentiometer P_1 to set I_{LOOP} at 10 mA. At this operating point, $I_C=4$ mA, and the balance of the current ($I_p=6$ mA) goes to the shunt regulator, the op amp, and the sensor-circuit load. The maximum load current is typically 5 mA. As V_{IN} increases from 2 to 10V, I_{LOOP} increases proportionally, solely from the increase in I_p, because I_C is constant:

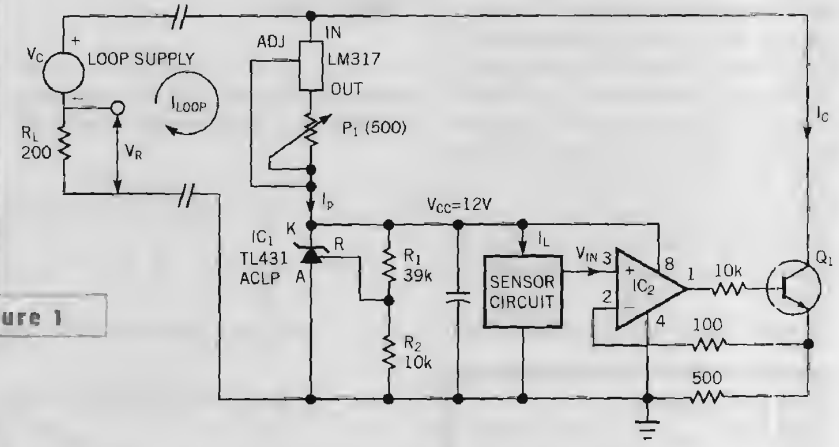


Figure 1

Inexpensive components provide an efficient loop-powered current transmitter.

V_{IN}	V_R
2V	2V
3V	2.4V
4V	2.8V
6V	3.6V
8V	4.4V
10V	5.2V

For every 1V increase in V_{IN}, the change in V_R is 0.4V. The circuit has good accuracy and temperature stability and operates with loop supplies of 20 to 30V. The maximum available sensor-circuit current is approximately 5 mA. Currents greater than this value result in poor V_{CC} regulation. (DI #2421).

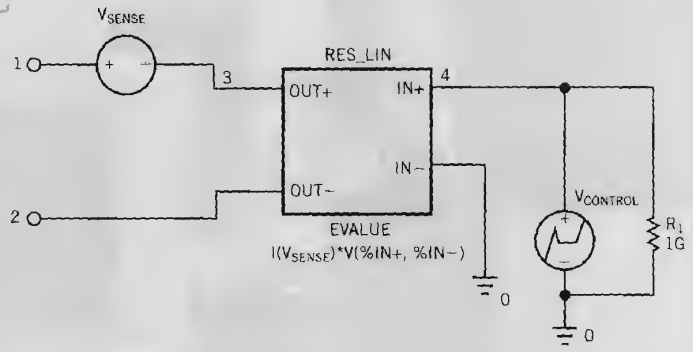
To Vote For This Design,
Circle No. 359

Spice creates time-variant resistor

Vittorio Ricchiuti, Italtel, L'Aquila, Italy

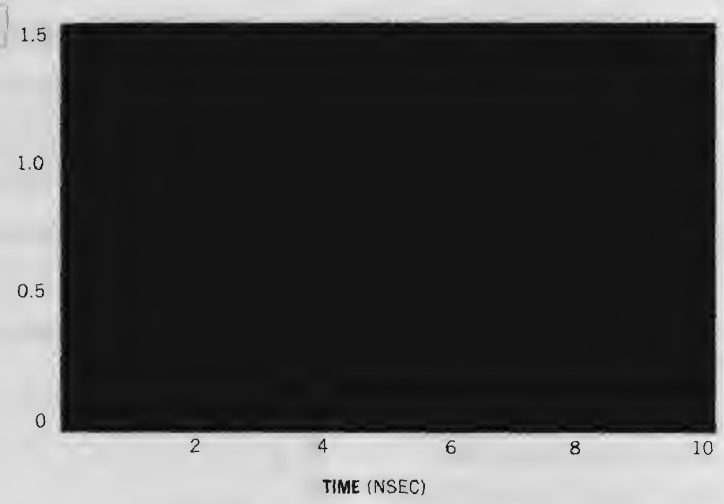
FIGURE 1 SHOWS A SUBCIRCUIT that represents the behavioral model of a linearly time-variant resistor in PSpice (Orcad, www.orcad.com). PSpice libraries include no models of such resistors, which you can use in various applications—for example, to simulate the fallback characteristic of a linear regulator in transient analysis. The suggested model uses the E-device in the PSpice analog-behavioral-modeling parts library. **Listing 1** describes the subcircuit in **Figure 1** for a time-variant resistor with the characteristic in **Figure 2**. The function in **Figure 2** is the product of an independent voltage source, $V_{CONTROL}$. The independent zero-voltage source V_{SENSE} senses the current $I(V_{SENSE})$ through the resistor. The voltage V_R between nodes 1 and 2 of the time-variant resistor is $V_R = f(t)*I(V_{SENSE})$, where f(t) is a function of $V_{CONTROL}$. (DI #2424)

Figure 1



A subcircuit provides the behavioral model of a time-variant resistor.

Figure 2



The routine in Listing 1 simulates a linearly time-variant resistor.

To Vote For This Design, Circle No. 360

LISTING 1—MACRO MODEL OF TIME-VARIANT RESISTOR

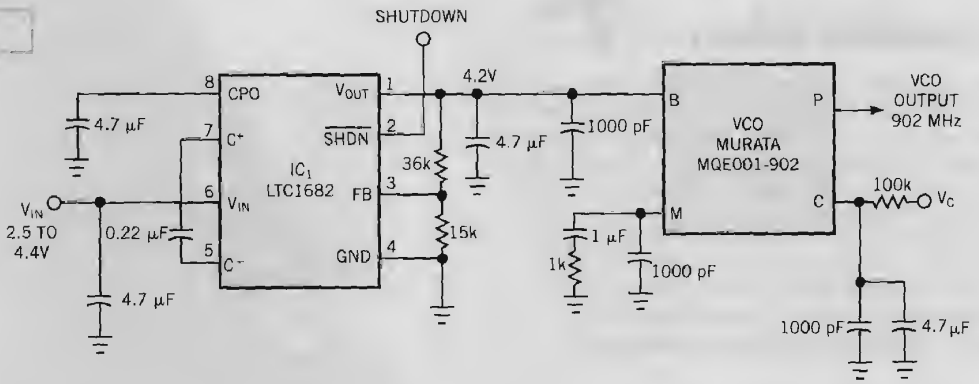
```
*
.SUBCKT    VAR_RES 1 2
*
R_R1       0 4 1G
*
V_Vcontrol 4 0
+PWL 2n 1k 4n 0 6n 1k
*
E_Res_lin  3 2 VALUE { I(V_Vsense)*V(4, 0) }
*
V_Vsense   1 3 DC 0V AC 0V 0V
*
.ENDS      VAR_RES
```

VCO supply touts low noise

Ted Henderson, Linear Technology Corp, Milpitas, CA

MANY PORTABLE RF PRODUCTS use VCOs to generate the RF carrier frequency. These applications often require low-noise VCO power-supply voltages that are higher than the primary battery voltage. Many designs use a dc/dc converter powering a low-noise linear regulator. This approach has several inherent disadvantages. The dc/dc converter produces noise that the regulator may not reject, resulting in regulator-output noise far greater than the thermal-noise

Figure 1



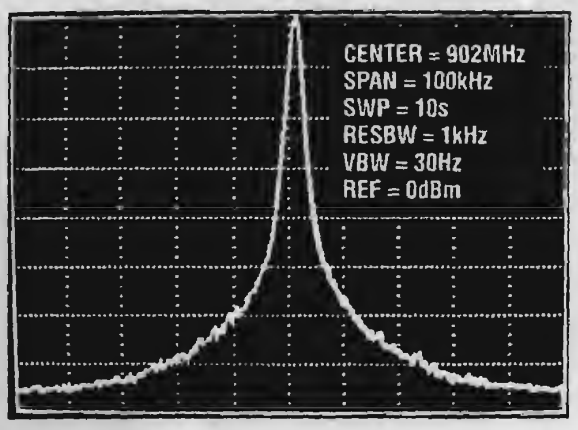
A single IC can generate a low-noise voltage for controlling a VCO.

levels. The linear regulator might require a large output-compensation capacitor with stringent ESR requirements. Finally, the board area for both devices and support components can be large. The circuit in **Figure 1** operates from an input range of 2.5 to 4.4V and generates a 4.2V low-noise voltage for a 900-MHz VCO.

The circuit is based on IC_1, an LTC1682 charge-pump dc/dc converter. The device includes a charge-pump/linear-regulator tandem that's optimized for minimum regulator-output noise. The linear regulator operates with several types of output capacitors, including small, low-value, low-ESR ceramic capacitors. **Figure 2** shows the close-in phase noise of the VCO operating in open-loop mode; **Figure 3** shows the peak-to-peak noise voltage at the regulator's output. (DI #2430).

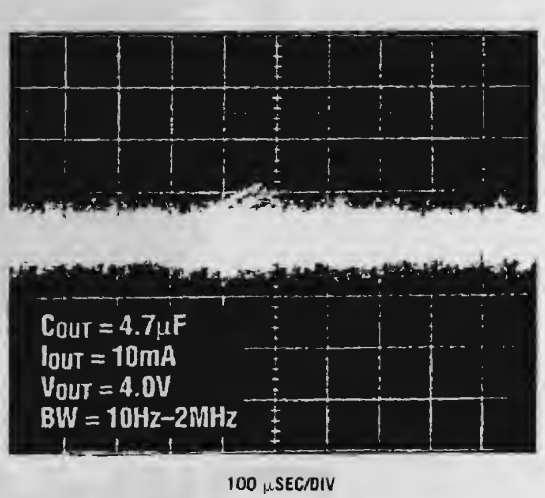
To Vote For This Design,
Circle No. 361

Figure 2



The VCO in Figure 1 exhibits low close-in phase noise.

Figure 3



The supply in Figure 1 generates only approximately 200 mV of peak-to-peak noise.

µC forms FM oscillator

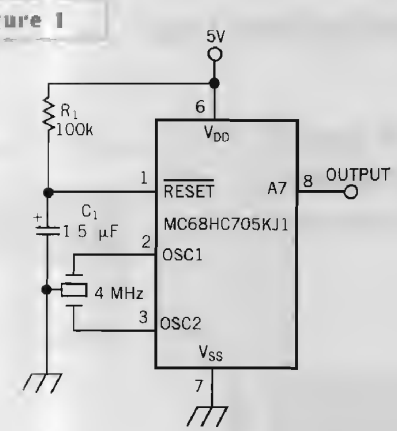
Abel Raynus, Armatron International, Melrose, MA

A PROJECT REQUIRED AN inexpensive oscillator whose frequency increased step by step from 200 to 400 Hz and then decreased to 200 Hz. The first step was to design a VCO with a staircase driver. However, this approach entailed at least four ICs and many discrete components. An alternative method (**Figure 1**) requires only one 16-pin µC (an MC68HC705KJ1, costing less than $1) and only a few external components. The process takes place exclusively in software (**Listing 1**). The µC generates a burst of 256 cycles of a given period. After that, the period decrements or increments, and the µC generates a burst with the new period. The choice of decrement or increment depends on the contents of the flag register. If the flag is zero, it is a sign to decrement; otherwise, to increment. When the set of frequencies reaches completion, the content of the register inverts. So, in one operation, the frequencies grow; in the next, they decrease.

Keep in mind that a linear change in the value of the period causes a nonlinear change in the frequency value (according to a hyperbolic curve). So, you should perform all calculations in the time domain, rather than in the frequency domain. In our application, the nonlinear aspect of the frequency is unimportant. However, if necessary, you can realize any kind of FM by organizing the table of frequencies corresponding to each step. The design determines the period resolution, ΔT. In this case, it equals 0.02 msec, because this value allows you to use only one 8-bit register (HPER) to perform all period-changing operations

For more precision, you could use two or more registers. The number of frequencies in the set is $K2_{MAX}=(T_{MAX}-$

Figure 1



An inexpensive µC replaces complicated VCO circuitry to configure a simple frequency-step generator.

LISTING 1—µC ASSEMBLY CODE FOR FREQUENCY GENERATOR

```
                    1  * Quasy FM Oscillator
                    2  * range 200 Hz - 400 Hz,
                    3  * period discretion - 0.02 ms
                    4  * number of frequencies - 125
                    5  ************************************************
                    6  *nolist
0000                7  $include "std-j1a.asm"
                    8  *list
                    9  *I/O PORTS
0000               10  out     equ   7;prtA pin11
                   11  *CONSTANTS
0000               12  HPmax   equ   250T; frequency = 200 Hz
                   13  *VARIABLES
00C0               14          org   RAM
00C0               15  k1      rmb   1    ;sound duration counter
00C1               16  k2      rmb   1    ;frequency number counter
00C2               17  HPER    rmb   1    ;Half of Period register
00C3               18  flag    rmb   1    ;flag register
                   19  *INITIALIZATION
07F1               20          org   MOR
07F1 20            21          fcb   %00100000 ;Osc.parall.resitor
0300               22          org   ROM
0300 A6FF          23  init    lda   #$ff
0302 B704          24          sta   ddrA ;set prtA for output
0304 3FC0          25          clr   k1   ;0 -> k1
0306 3FC1          26          clr   k2   ;0 -> k2
0308 3FC3          27          clr   flag
030A A6FA          28          lda   #HPmax
030C B7C2          29          sta   HPER ;set T = 5 ms, f = 200 kHz
030E CD032D        30  main    jsr   burst; generate 256 cycles
0311 3DC3          31          tst   flag
0313 2714          32          beq   m1
0315 3CC2          33          inc   HPER ; HPER + 1
0317 3CC1          34  m2      inc   k2   ; k2 + 1
0319 B6C1          35          lda   k2   ; k2 -> ACC
031B A17D          36          cmp   #125T; k2 < 125?
031D 25EF          37          blo   main
031F B6C3          38          lda   flag ;inverting flag
0321 A801          39          eor   #1   ;
0323 B7C3          40          sta   flag ;
0325 3FC1          41          clr   k2   ; 0 -> k2
0327 20E5          42          bra   main
0329 3AC2          43  m1      dec   HPER ;HPER - 1
032B 20EA          44          bra   m2
032D 1E00          45  burst   bset  out,prtA  ;1 -> out
032F BEC2          46          ldx   HPER      ;HPER -> x
0331 CD0342        47          jsr   dly01x    ;half period delay
0334 1F00          48          bclr  out,prtA  ;0 -> out
0336 BEC2          49          ldx   HPER      ;HPER -> x
0338 CD0342        50          jsr   dly01x    ;half period delay
033B 3CC0          51          inc   k1        ;k1 + 1
033D 3DC0          52          tst   k1        ;k1 = 0?
033F 26EC          53          bne   burst
0341 81            54          rts             ;return from burst
0342 A602          55  dly01x  lda   #2        ;delay 0.01x ms
0344 4A            56  rep0    decA
0345 26FD          57          bne   rep0
0347 5A            58          decx
0348 26F8          59          bne   dly01x
034A 81            60          rts             ;return from dly01x
                   61  ************************************************
07FE               62          org   VECTORS+6
07FE 0300          63          fdb   init
```

T_{MIN})/ΔT. You can determine any frequency is the set as:

Flag =0; $f_K = 1/(T_{MAX} - \Delta T \cdot K2)$.

Flag=1; $f_K = 1/(T_{MIN} + \Delta T \cdot K2)$.

Table 1 shows some of the frequencies. The duration of the set is $t = \frac{1}{2}(T_{MAX} + T_{MIN}) \cdot K1_{MAX} \cdot K2_{MAX}$. In this case, t=2 msec (DI #2432).

To Vote For This Design, Circle No. 362

TABLE 1–200- TO 400-Hz FREQUENCY STEPS

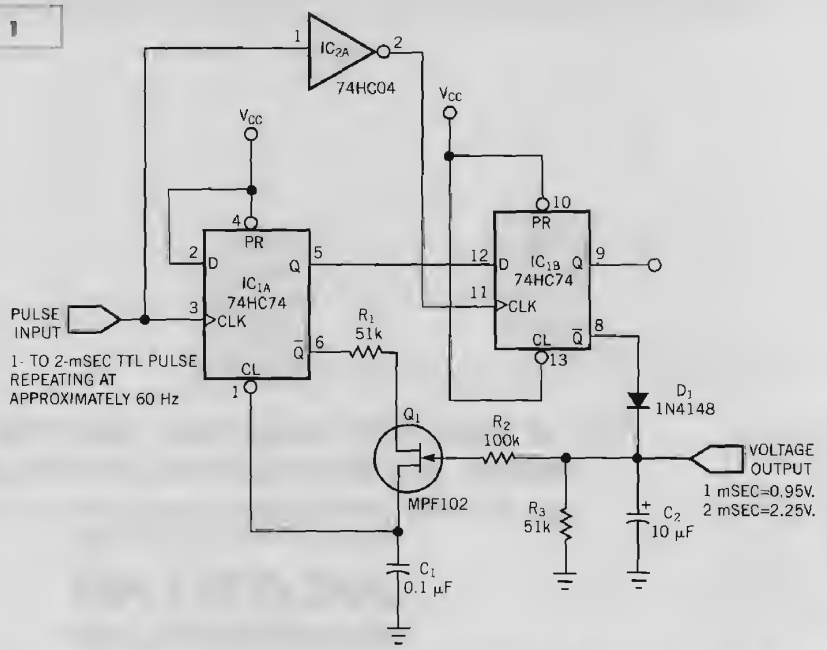
Flag	K2	T (msec)	½T (msec)	HPER	F (Hz)	Comments
0	0	5.00	2.50	250	200.0	T_{MAX}
0	1	4.98	2.49	249	200.8	
0	2	4.96	2.48	248	201.6	
0	3	4.94	2.47	247	202.4	
...	...	...	...	...	...	
0	123	2.54	1.27	127	393.7	
0	124	2.52	1.26	126	396.8	
0	0	2.50	1.25	125	400.0	T_{MIN}
1	1	2.52	1.26	126	396.8	
1	2	2.54	1.27	127	393.7	
1	3	2.56	1.28	128	390.6	

Circuit converts pulse width to voltage

Ron Hegler, Picker International, Cleveland, OH

The circuit in **Figure 1** stems from a radio-controlled modeling application, which requires a voltage proportional to the width of the incoming servo pulses. The circuit is optimized for a positive-going pulse width of 1 to 2 msec, repeating at intervals of approximately 17 msec. The output produces a voltage of 0.95V for a 1-msec pulse to 2.25V for a 2-msec pulse. The circuit operates similarly to a PLL, but it locks onto the pulse width, rather than to the frequency, of the incoming signal. IC_{1A} is a one-shot multivibrator with its time constant a function of R_1, the FET's on-resistance, and C_1. IC_{1B} is a pulse-width comparator that compares the reference pulse from IC_{1A} with the incoming pulse. Upon the rise of the incoming pulse, IC_{1A}'s Q output clocks high and drives IC_{1B}'s D input high. If IC_{1A} times out before the input pulse falls, IC_{1A}'s Q output goes low, driving the D input of IC_{1B} low. This action drives the $\overline{Q}$ of IC_{1B} high when the input pulse falls. The $\overline{Q}$ output connects to the FET through D_1 and R_2. C_2 filters the $\overline{Q}$ output to adjust the time constant of one-shot IC_{1A} to match the incoming pulse. The voltage across C_2 indicates the incoming pulse width. (DI #2431).

Figure 1



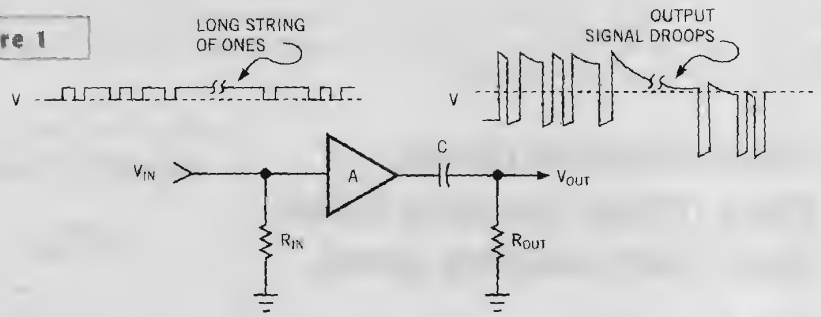
This pulse-width-locked loop provides an output voltage proportional to the incoming pulse width.

To Vote For This Design, Circle No. 363

Restore dc to NRZ sigmals

Jay Kirschenbaum, Columbia University, New York, NY

Figure 1

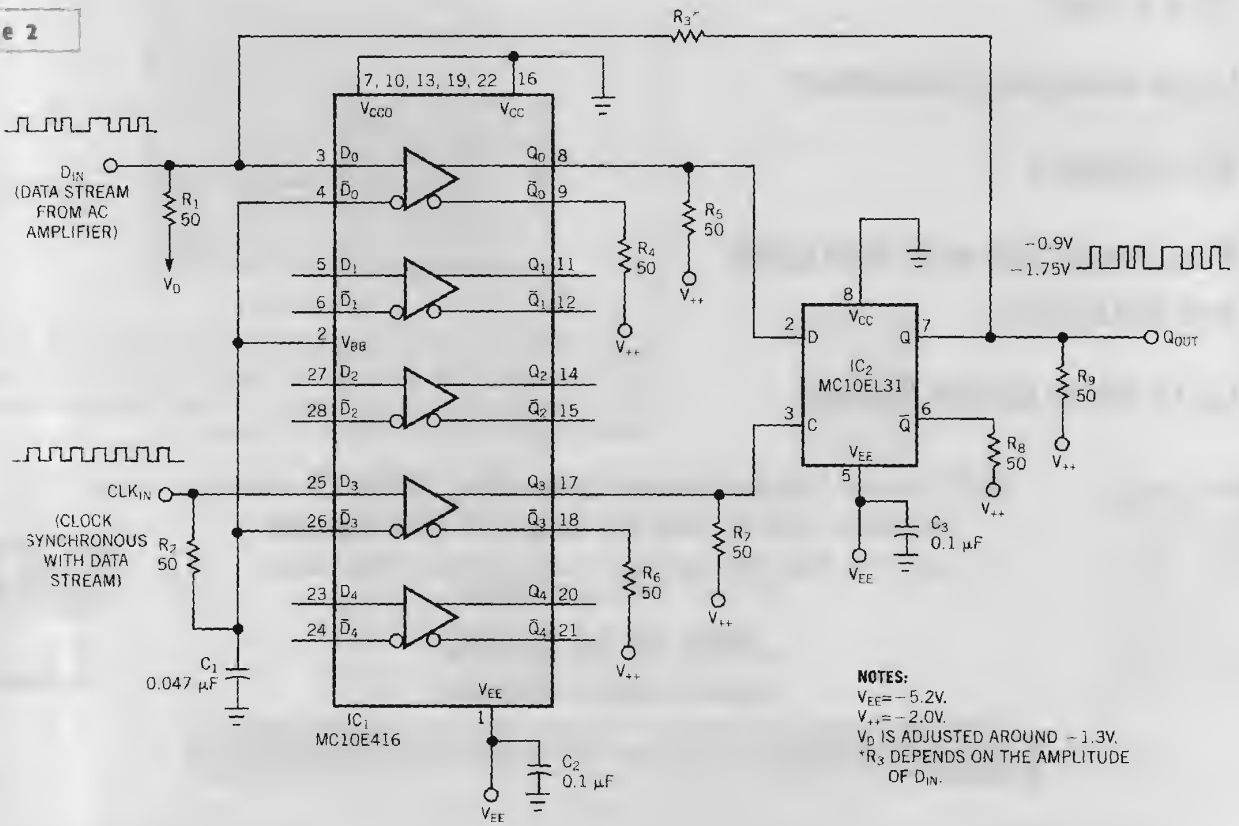


A long string of ones or zeros leads to droop in a purely ac-coupled digital amplifier.

For NRZ signals, ac amplification is preferable to dc amplification, because ac amplification is usually more economical and has greater immunity to drift. However, ac processing has the disadvantage that ac coupling removes the dc reference level of the digital data stream. As **Figure 1** shows, if a long stream of ones or zeros appears at the amplifier's input, the output droops after a period determned by the RC time constant at the input. This problem makes it necessary to include a dc-restoration circuit following the ac amplifier.

The circuit in **Figure 2** restores the dc level that the amplifier removes and eliminates droop, even if long strings of ones or zeros appear in the data stream. The circuit also provides additional amplification, increasing the signal to ECL levels. Synchronous clocking gives the circuit good immunity to noise in the data channel. Tests reveal that the circuit op-

Figure 2



This circuit eliminates the droop from the data stream in Figure 1 and simultaneously amplifies the digital signal to ECL levels.

erates satisfactorily at clock speeds from dc to 1 GHz.

IC_1 is an MC10E416 five-channel line receiver, which boosts digital signals from a minimum of 50 mV p-p to ECL levels. The input circuit of each channel of IC_1 is a differential amplifier, which responds when the signal level at the true input swings past the signal at the complementary input. In **Figure 2**, the input data stream goes to the D_0 channel. The complementary input $\overline{D}_0$ connects to the V_{BB} pin, which is at −1.3V. The true input D_0 connects to bias voltage V_D through 50Ω. V_D is also nominally −1.3V, but you can tweak it to compensate for device characteristics. IC_2 is an MC10EL31 flip-flop. Both ICs are specified to 2 GHz. The logic level at D_{IN} transfers to Q_{OUT} at the positive transition of the signal at CLK_{IN}. Resistor R_3 and the 50Ω pull-down resistor, R_1, form a voltage divider that puts D_0 above or below $\overline{D}_0$ by half the expected input-voltage swing. The value of R_3 is a function of the swing of the input signal. For example, assume D_{IN} has the value 100 mV p-p. If R_3 is 370Ω, and Q_{OUT} is at logic zero (−1.75V), the voltage divider sets D_0 50 mV below $\overline{D}_0$, and Q_{OUT} stays at logic zero as long as these conditions prevail.

Now suppose that logic one appears at D_{IN}. This condition simultaneously puts D_0 50 mV above $\overline{D}_0$, and Q_{OUT} switches to logic one (−0.9V) at the next clock transition. When this condition happens, the R_1-R_3 voltage divider holds D_0 50 mV above $\overline{D}_0$, and Q_{OUT} stays at logic one until the next transition to logic zero. Voltage droop disappears, and the dc reference of the digital signal re-establishes itself. Moreover, the circuit amplifies the 100-mV swing of the digital signal to ECL levels. Because the circuit triggers only when the clock edge appears, it's highly immune to noise in the data channel. At power-up, one transition from logic zero to logic one and one transition from logic one to logic zero are necessary at the amplfier input to put the circuit in a defined state. After that sequence, the circuit faithfully follows the input.

Proper timing between the data and the clock is important to reach the maximum operating speed of the circuit. Ideally, the clock edge should arrive at IC_1 as soon as the data-setup time is over, switching Q_{OUT} and pulling up D_{IN} as close to the data transition as possible. The theoretical upper limit on the operating (clock) frequency is a function of the propagation delay through the circuit (825 psec at 25°C). Data transitions must be spaced at least this far apart, so the maximum clock frequency is 1.2 GHz. Tests show that the circuit operates successfully at clock frequencies of dc to 1 GHz and data frequencies of dc to 500 MHz. The tests used a square wave for the clock input and a synchronous square wave of a lower frequency at the data input. The circuit restores the dc level and amplifies the signal to ECL levels over the entire test range. (DI #2434).

To Vote For This Design, Circle No. 364